Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Applied Mathematics 154 (2006) 754–769

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Noising methods for a clique partitioning problem☆

Irène Charon, Olivier Hudry

École nationale supérieure des télécommunications, 46, rue Barrault, 75634 Paris cedex 13, France

Received 24 November 2002; received in revised form 19 May 2004; accepted 31 May 2005
Available online 1 December 2005

Abstract

This paper deals with the application of noising methods to a clique partitioning problem for a weighted graph. The aim is to study different ways to add noise to the data, and to show that the choice of the noise-adding-scheme may have some impact on the performance of these methods. Among the noise-adding-schemes described here, two of them are totally new, leading to the "forgotten vertices" and to the "forgotten edges" methods. We also experimentally study a generic noising method that automatically tunes its parameters. For each noise-adding-scheme, we compare a variant which inserts descents and a variant which does not.

Keywords: Noising methods; Metaheuristics; Simulated annealing; Threshold accepting algorithms; Clique partitioning of a weighted graph; Aggregation of relations; Classification; Clustering; Régnier's problem; Zahn's problem

1. Introduction

In 1993, we designed a new metaheuristic for discrete optimization problems: the *noising method* [7], and then we generalized it into a family of methods (mainly depending on the noise-adding-scheme but also on other components; see [10] for a review of the principles and of the applications of the noising methods). As other metaheuristics (for references, see for instance [27–29,1,32,24,19]), the noising methods are not designed to solve only one specific problem, but different kinds of combinatorial optimization problems. Such a problem can be described as follows:

$$\text{minimize } f(s) \text{ for } s \in S,$$

where S is the finite solutions set, and $f(s)$ gives the value of the solution s. Note that sometimes S is not given explicitly. Then, we assume that the solutions can be generated in a reasonable time (it will the case for the problem addressed in this paper).

The noising methods are based on *elementary* (or *local*) *transformations*. An *elementary transformation* is an operation which, when applied to a solution s of S, changes s into another solution s' of S by modifying one (or some) feature(s) of s without changing its global structure. For instance, if s is a binary string, an elementary transformation may consist in changing one bit into its complement. The new solution s' is called a *neighbor* of s. More generally, the *neighborhood* $N(s)$ of s is the set of solutions generated by applying the elementary transformation to s. For instance,

☆ This study has been supported by the French "Centre national de la recherche scientifique" (CNRS), inside the research project number 0693 (genomics) entitled "Nouvelles méthodes de partitionnement pour la classification d'objets biologiques utilisant des données structurales ou des données d'interactions fonctionnelles". We would like to thank Alain Guénoche (CNRS, Marseille) for having associated us with him in this project.
E-mail address: hudry@infres.enst.fr (O. Hudry).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.05.029

if s is still a binary string of n bits and if the elementary transformation is the one evocated previously, $N(s)$ is a set of n binary strings: the n binary strings generated by changing one of the n bits of s. (See the above references for more details about these basic definitions.) Of course, if we want to be able to explore the whole set S of solutions, and not to be restricted to only a part of it, a desirable property of the transformation is that, when repeated suitably, this transformation could lead from any element of S to any other one, or at least to an optimal solution.

Based on a given elementary transformation (or on a given neighborhood), we may design an *iterative improvement method*, also called a *descent method* (or simply a *descent* in the sequel), or sometimes a *quenching*, for a minimization problem. In a descent, we start from an initial solution s_0 (for example randomly generated) and we generate a sequence of solutions $s_1, s_2, \ldots,$ until a solution s_q (where q is not known a priori) is reached with the following properties:

1. $\forall i \in \{1, 2, \ldots, q\}, s_i \in N(s_{i-1})$;
2. $\forall i \in \{1, 2, \ldots, q\}, f(s_i) < f(s_{i-1})$;
3. $\forall s \in N(s_q), f(s) \geqslant f(s_q)$.

Property 3 means that a descent stops when a local (with respect to the adopted elementary transformation or to the adopted neighborhood) minimum has been found.

The noising methods are based on the same principles but, instead of the genuine function f to minimize, we consider that f has been perturbed by *noises*. Thus, as for a descent, the noising methods apply elementary transformations; but to know whether such a transformation is accepted or not, we take these noises into account. This involves that a "bad" transformation (that is, a transformation leading to an increase of f) can be accepted (as in simulated annealing for instance) but also that a "good" one (that is, a transformation leading to a decrease of f) can be rejected because of the noises (see Section 3 about how to add these noises). In order to go back to the genuine function f to minimize at the end of the process, the range of the noises decreases during the run of the method, typically down to 0 (but it is often possible to stop before, as shown in [9]): then there is no added noise and we deal with f itself.

It is also necessary to precise the way to explore the neighborhood. There are different possibilities: for instance, the exploration is a random one in a classic simulated annealing while it is an exhaustive one in a classic tabu search. In the noising methods, we assume that the neighborhood is implicitly ordered (even if we do not know this order explicitly) and the neighbors (or rather the elementary transformations defining them) are scanned in this order, one after the other. Then we adopt the first neighbor which is better (with respect to the perturbed function) than the current solution. Notice that the implicit order is not necessarily the same for all the solutions, and even for a given solution, it may change from one iteration to another (thus, if we scan twice the neighborhood of a same solution, its neighbors are not necessarily ranked in the same order). This way of exploring the neighborhood is sometimes called a "cyclic exploration" [15].

The aim of this paper is to compare the performance of 18 variants of noising methods derived from six noise-adding-schemes and three variants for each noise-adding-scheme. One of the six noise-adding-schemes (called "basic" in the following) has already been applied to the problem considered in this paper ([7]; see also [21,22,31] for similar application). Three of them can be found in different papers dealing with noising methods, but were applied to other problems (see [10]); they include a scheme near the one of a classic simulated annealing and another one close to the one of the threshold accepting methods (by the way, note that De Amorim et al. [14], compared a classic simulated annealing and a tabu search to this problem: according to their experiments, both methods lead to qualitatively similar results for random graphs). The two others (called "forgotten vertices" and "forgotten edges") are totally new, in the sense that no paper dealing with them has been published yet. Similarly, the application of the "automatically tuned" variant is new (the principles of this automatic way of tuning the parameters of the noising methods can be found in [11]).

This study is done on a partitioning problem described below (Section 2); it arises in different contexts and it is also one of our aims to be able to solve this problem as accurately as possible, especially for its applications in genomics (see for instance [18,20]) where it is quite important to get close to an optimal solution within a "reasonable" CPU time. In Section 3, we detail the 18 noising methods. Experimental results can be found in Section 5. We discuss them in Section 6: broadly speaking, it appears that the new noise-adding schemes "forgotten vertices" and "forgotten edges" give very good results, better than those provided by the other schemes, and that the "automatically tuned" variant gives also very good results, almost as good as those that we compute with a sharp tuning of the parameters, while the user has nothing to do to tune these parameters. Section 6 is devoted to the global conclusions.

2. The clique partitioning problem

At the origin, the problem in which we are interested deals with the *aggregation and the approximation of symmetric relations into an equivalence relation*. More particularly, we pay attention to two problems: the first one, arising in cluster analysis and defined by Régnier [30], consists in the aggregation of equivalence relations into a unique equivalence relation; the second one, arising in social sciences and set by Zahn [35], consists in the approximation of a symmetric relation by an equivalence relation. In fact, these problems model other problems arising in different contexts: classification, psychometry, genomics, and so on (for references upon these problems and their applications, see for instance [2–4,23,18,20]).

Régnier's problem arises for example in classification (clustering). In this problem, we deal with a set X of n objects and a set of m criteria; each criterion is assumed to define an equivalence relation on X; the aim is to find a unique equivalence relation defined on X which summarizes the m criteria as well as possible by minimizing the total number of disagreements with respect to the m criteria. This problem is NP-hard when m is not fixed, and its status is not known for a fixed value of m (see [2]).

Zahn's problem consists in approximating a given symmetric relation S defined on a set X by an equivalence relation E defined also on X which is at minimum distance to S with respect to the *symmetric difference distance* (which measures the number of disagreements between S and E). This problem is NP-hard too [26].

These two problems can be represented by the following *clique partitioning problem* (CPP in what follows; see [33,34] for details). In this CPP, we consider a weighted non-oriented graph $G = (X, U, w)$ with n vertices; this graph is complete; a positive or negative integer $w(x, y) = w(y, x)$ is associated with each edge $\{x, y\} \in U (x \neq y)$; our CPP consists in finding a partition of X into $k(G)$ cliques $C_1^*, C_2^*, \ldots, C_{k(G)}^*$ (hence the number $k(G)$ of cliques is not fixed a priori and depends on G, or more precisely on n and w; for this reason and because of the sign of the weights, CPP is not the well-known k-Cut problem), in order to minimize the sum of the weights of the edges with their two extremities in a same clique, that is the function f defined for any partition $(C_1, C_2, \ldots, C_k)$ of X by

$$f(C_1, C_2, \ldots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \sum_{\substack{(x,y) \in C_i \times C_i \\ x \neq y}} w(x, y).$$

This CPP is also NP-hard (see [33,34]). To formulate Régnier's problem and Zahn's problem as instances of CPP, we build a weighted complete graph as follows (see [33,34] details). The vertex set will be the set X of Régnier's or Zahn's problems. For Régnier's problem, the weight of an edge $\{x, y\}$ (with $x \neq y$) is given by the difference $m - 2m_{xy}$, where m_{xy} denotes the number of criteria for which x and y are together in a same class (this weight is also the difference between the number, equal to $m - m_{xy}$, of criteria for which x and y are not together in a same class, and m_{xy}); note that m_{xy} is an integer between $-m$ and m. For Zahn's problem, let S be the symmetric relation of the instance; the weight of an edge $\{x, y\}$ (with $x \neq y$) is -1 if x and y are in relation with respect to S and $+1$ otherwise. Then the search of an equivalence relation solution of Régnier's or Zahn's problems consists in both cases in partitioning the weighted complete graph into disjoint cliques in order to minimize the sum of the weights of the edges with their two extremities in a same clique; hence our partitioning problem. By the way, note that minimizing the innerclass weights as we do is the same as maximizing the outerclass weights (because the sum of these weights is a constant). Moreover, as the signs of the weights are not necessarily the same for all weights, CPP is also equivalent to the maximization of the innerclass weights or to the minimization outerclass weights (it is sufficient to multiply all the weights by -1).

3. The noising methods applied to the clique partitioning problem

To apply the noising methods to the clique partitioning problem, we must first precise the elementary transformation used to define the neighborhood. In our work, it consists in moving a vertex from its current class to another one, which can be empty if we want to create a new class. Below, when we speak about "each current class C", we mean in fact "each class of the current solution, including the empty class", and when we consider the possibility to move a vertex from its current class to another one, this one can be the empty class. This transformation was already suggested by Régnier [30]. Note that the size of the neighborhood of a solution is not always the same during the run of the algorithm:

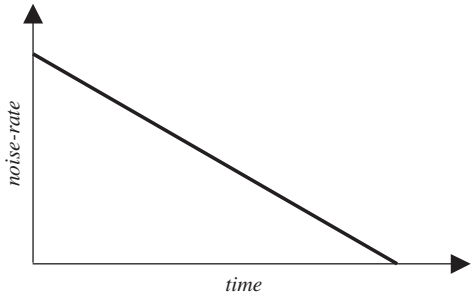

Fig. 1. Decreasing of the noise-rate for the variant B.

it is equal to $(k+1)n-\alpha$, where k denotes the number of classes (which is not fixed) of the current solution, n the number of vertices and α the number of vertices alone in their classes.

The exploration of the neighborhood is done as follows.

In a descent, to look for and to choose an improving elementary transformation, we consider the vertices one by one. For each vertex x, we compute the best class for x among the classes of the current solution. More precisely, for any class C of the current solution, we set $W(x,\emptyset)=0$, and, for $C\neq\emptyset$, $W(x,C)=\sum_{\substack{y\in C\\ y\neq x}} w(x,y)$. It is easy to see that moving x from its current class C_x to another one C involves a variation for f of the quantity $W(x,C)-W(x,C_x)$; thus moving x from C_x to C leads to an improvement if $W(x,C)<W(x,C_x)$. The class of the current solution in which it is better to put x (when nothing else is changed) is called the *best class* of x; it is the class C^* for which $W(x,C^*)$ is minimum (notice the inequality $W(x,C^*)\leqslant 0$ since $W(x,\emptyset)=0$). So, the complexity of computing the best class of any vertex can be done in $O(n)$.

From the current solution, when we discover that a vertex is not in its best class, we move it towards its best class and so the current solution changes; then we apply the same process to the new current solution. The exploration of the neighborhood is a "cyclic" one (see [15]): initially, the vertices are ranked in a random order generated uniformly; all the vertices are tried once before any is considered for a second time and an improving transformation is adopted as soon as it has been discovered; this process is applied until a complete cyclic exploration of the neighborhood has been completed without finding any possible improvement; then all the vertices are in their best classes (remember that it means only that moving any single vertex from its current class to another one does not lead to a better solution) and the current solution is a local minimum with respect to the neighborhood generated from the adopted elementary transformation.

When noises are added (in accordance with the schemes described below), the search follows the same features, but with respect to the perturbed quantities W. Moreover, because of the noises, we cannot adopt the same criterion to stop a noising method as for a descent, and thus it is necessary to fix the number of "perturbed iterations"; this number is given by the user or is computed automatically for the variant A (for "automatic"; see below) according to the CPU time that he or she wishes to spend in order to solve his or her problem. At the beginning of each cyclic exploration of the neighborhood, the vertices are ranked in a new random order before being scanned.

We may now describe the different noising methods that we study in this paper. They are based on different types of noise-adding-schemes and, for each type, we consider three variants.

3.1. Variant B

In the first variant, the noise rate always decreases linearly down to 0. The name of the methods following this scheme will always end with B (B stands for "basic"). Fig. 1 illustrates the decreasing of the noise rate for this variant B.

3.2. Variant D

In the second variant, the noise rate decreases also linearly but "unperturbed" descents are inserted from time to time during the noise-adding process; the name of these methods will always end with D. From our experiments on the partitioning problem and on other combinatorial optimization problems (see [10]), it appears that performing four

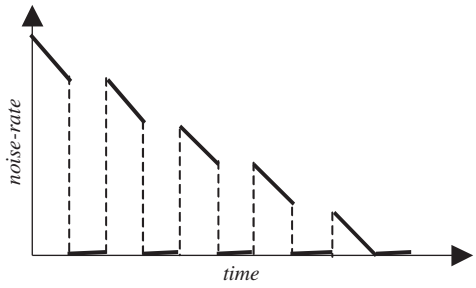

Fig. 2. Decreasing of the noise-rate for the variant D.

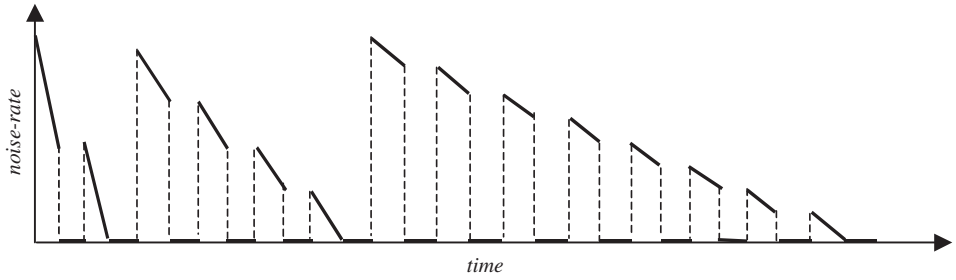

Fig. 3. Evolution of the noise-rate for the variant A.

cyclic explorations of the neighborhood with respect to the perturbed quantities W between two (unperturbed) descents seems to be a good choice. So it is what we do here. It means that, between two descents, each vertex is scanned four times in order to find its best perturbed class in the current solution. Fig. 2 illustrates the decreasing of the noise rate for the variant D: when a descent is performed, it is in fact the same as applying a noising method with a noise rate equal to 0.

3.3. Variant A

The third variant is an automatically tuned version of the noising methods with the insertion of unperturbed descents as in the second variant; the name of these methods will always end with A (for "automatic"). The goal of these variants is to find appropriate values for the initial noise rate and for the number of iterations which will be performed. Thus, the only parameter given by the user in these variants is the CPU time that he or she wishes to spend for his or her problem. It is a generic automatic method which can be applied with various noises or to various problems by changing only the type of noise or the characteristics defining the problem; no parameter of this generic method is changed from one noise-adding-scheme to another one or from a problem to another one. The principle of this automatically tuned variant consists in applying a noising method several times, in the sense that the noise rate decreases several times from a maximum rate down to 0. The duration of each noising method is twice the one of the noising method applied just before the current one, while the first noising method is very short. The aim of this repetition of noising methods is to compute a suitable initial noise rate: this one is improved during the process. We cannot depict here all the details of these automatically tuned noising methods, but the interested reader will find them in [11] (in which we study the behavior of this automatic variant of the noising methods when applied to several combinatorial optimization problems, including the well-known Traveling Salesman problem). Fig. 3 illustrates the evolution of the noise rate during the run of this variant.

In the noising methods, noises are sometimes added to the data, or directly to the variations of f when we consider a neighbor of the current solution; they can be added in an absolute or a relative way; they can call on a random variable following a uniform law, or a Gaussian one, or another one (see [10]). In addition to these possibilities, we consider here a new way to design noising methods (it appeared in 1999, see [8] similar ideas can be found in [6]); it consists in

"forgetting" a part of the data (here, vertices or edges of the considered graph); this forgotten part decreases during the run of the method in order to deal with all the data in the last iterations…It corresponds anyway to different probability laws to know whether a transformation is accepted and these laws are always computed from something which can be considered as a noise. It is worth noticing that, at least from a theoretical point of view, these different ways of perturbing f can be linked together, as shown in Charon and Hudry [10]; anyway, in this case, it is not always easy to find the equivalent probability laws which govern the choices of the noises.

We describe below how to choose the noises. As said above, when we are looking for the next elementary transformation to apply, we consider the vertices one by one; for each vertex x, we compute its best perturbed class, that is the class C^* which corresponds with the minimum value of a quantity denoted below by $W_p(x, C)$ over the classes of the current solution and which depends on the added noises; then x is moved into its best perturbed class if x does not belong already to this class in the current solution.

For each type of noise, we have three methods according to the three variants: basic (letter B), with descents (letter D), automatic (letter A). In all the methods that we are going to describe, there is a variable called *rate* (the rate of noise, or the rate of forgotten vertices, or the rate of forgotten edges) which decreases linearly during the process (or during the different parts of the process for automatic methods) from a maximum rate *max_rate* down to 0.

3.4. Uniform noise: methods UB, UD, UA

In the methods UB, UD and UA (U for "uniform"), for a vertex x and for each current class C, $W_p(x, C)$ is given by the sum of $W(x, C)$ and a noise $\rho \times rate$ where ρ is randomly chosen in the interval $[-1, 1]$ with a uniform law at each time that we consider x and C:

$$W_p(x, C) = W(x, C) + \rho \times rate.$$

3.5. Logarithmic noise: methods LB, LD, LA

In the methods LB, LD and LA (L for "logarithmic"), for a vertex x and for each current class C, we randomly draw a number ρ in the interval $]0, 1]$ with a uniform law and we set:

$$W_p(x, C) = W(x, C) + rate \times \log(\rho).$$

There is a relationship between this noise and simulated annealing (see [10] for more details), since we have the equivalence:

$$W_p(x, C) < 0 \quad \Leftrightarrow \quad \rho < \mathrm{e}^{-W(x,C)/rate}.$$

Then, the noise rate *rate* used here can be considered as the usual temperature of simulated annealing. Among these variants, LB gives the scheme of a classic simulated annealing but with a cyclic exploration of the neighborhood (which is quite better, at least for this problem, than a random exploration; see [7], and more generally see references in [10]). Anyway, for homogeneity's sake, we go on to call it LB (instead of SA for instance).

3.6. Relative uniform noise: methods RB, RD, RA

In the methods RB, RD and RA (R for "relative"), noises modify the weights of the edges with the following pattern. For a vertex x and for each current class C, we set:

$$W_p(x, C) = \sum_{\substack{y \in C \\ y \neq x}} w(x, y) \times (1 + rate \times \rho_y),$$

where ρ_y is chosen randomly, with a uniform law, in the interval $[-1, 1]$.

3.7. Threshold methods TB, TD, TA

The methods TB, TD and TA (T for "threshold") come from the threshold accepting method proposed by Dueck et al. [16,17] (see also [10]; for details on the relationship between the threshold accepting method and the noising methods). For a vertex x, we compute all the values $W(x, C)$ and we select all the classes C verifying:

$$W(x, C) < W(x, C_x) + rate,$$

where C_x denotes the current class of x. The class in which x is moved is chosen randomly with a uniform distribution among all the selected classes.

3.8. Forgotten vertices methods FVB, FVD, FVA

In the methods FVB, FVD and FVA (FV for "forgotten vertices"), at the beginning of each cyclic exploration of the neighborhood, we randomly select vertices which will be forgotten during this exploration. The number of forgotten vertices at each exploration is given by the product of *rate* by the number of vertices of G. Clearly, the maximum rate, *max_rate*, cannot be higher than 1. In our experiments, we tried the values 0.1, 0.2, ..., 0.8 and 0.9 for *max_rate*; we report below only the best results that we obtained.

When we perform the cyclic exploration of the neighborhood, we do not search the best perturbed class for a forgotten vertex and, for a not forgotten vertex x and for each class C, we compute:

$$W_p(x, C) = \sum_{\substack{y \in C \\ y \neq x,\ y \text{ not forgotten}}} w(x, y),$$

and the best perturbed class of x is the class for which this value is minimum.

3.9. Forgotten edges methods FEB, FED, FEA

The methods FEB, FED and FEA (FE for "forgotten edges") are also new. They proceed as FVB, FVD and FVA but, instead of forgetting vertices, they forget edges. Then, for each vertex x and for each class C of the current solution, we compute:

$$W_p(x, C) = \sum_{\substack{y \in C \\ y \neq x,\ \{x,y\} \text{ not forgotten}}} w(x, y).$$

The probability that a given edge $\{x, y\}$ is forgotten is equal to *rate*: at each time that an edge is considered, we choose a number ρ in the interval [0, 1] randomly, with a uniform law; if ρ is less than *rate*, the edge is forgotten.

As for FVB, FVD and FVA, we tried the values 0.1, 0.2, ..., 0.8 and 0.9 for the maximum rate *max_rate* of forgotten edges; here also, we report below only the best results that we obtained.

4. Experimental results

The experiments that we report here have been done on seven types of graphs (they can be found at the URL http://www.infres.enst.fr/~charon/partition/); for each type, we did many tests on many graphs: we obtained the same qualitative conclusions over the different tests and the different graphs for each type; for this reason, we detail here the results obtained for only one graph of each type. Anyway, once again, the qualitative conclusions are corroborated by many other experimental results. The weights of the studied graphs are always integers. The results given below are averages on $N = 100$ trials, except for the graph called *rand100-100*, for which we performed $N = 1000$ trials. In each case, the initial solution is randomly computed. For a same graph, each method has been given the same CPU time. The time limits are set in such a way that, according to our experiments, nothing noticeable would happen after this limit. For the methods which depend on parameters, we tried to find the best parameters; the reported results have been obtained from these "best" parameters. Note that previous experiments done on the Traveling Salesman Problem

Table 1
Characteristics of the 19 studied methods

	Basic	With descents	Automatic tuning
Repeated descents		Q	
Uniform noise	UB	UD	UA
Relative noise	RB	RD	RA
Logarithmic noise	LB	LD	LA
Threshold noise	TB	TD	TA
Forgotten vertices	FVB	FVD	FVA
Forgotten edges	FEB	FED	FEA

[9] show that it is not necessary to tune the parameters of a noising method very sharply (it is why we may try to tune the noising methods automatically): for instance, tuning the initial noise rate at 10% or even 20% still gives very good results (better than those provided by a classic simulated annealing for example).

We also performed repeated descents (*quenching*) from random initial solutions during the same CPU time and kept the best value obtained by these descents; this repetition of descents is called "method Q" below. In this method, there is only one parameter: the number of descents which are repeated; this number is fixed so that the CPU time of Q is similar to the CPU time devoted to each one of the other methods. Otherwise, the features of Q are the same as for the other methods (same elementary transformation, same way of exploring the neighborhood; of course, the variants A, B, D are not relevant for Q).

For some methods and some graphs, the results that we obtained are not reported in our graphics, because, if we put them, the ordinates of the other results are too close one to the other and so it is difficult to see any difference between them; thus, these results are given in a table.

It is well-known that, when the number N of trials is great, the random variable associated with the average of the results provided by a method over these trials tends to follow a Gaussian law of parameters m_0 and $\sigma_0/\sqrt{N}$, where m_0 and σ_0 are respectively the average and standard deviation of the random variable associated with one experiment. The theoretic average that we would like to know is m_0. Because of this Gaussian law, m_0 can be located in the interval $[m - 1.96\sigma/\sqrt{N}, m + 1.96\sigma\sqrt{N}]$ with a probability equal to 0.95, where m and σ are the average and the standard deviation computed from the N trials: this gives the so-called "confidence interval at 95%". In the graphics below, for each one of our seven graphs and for each method, we have drawn segments indicating the confidence interval at 95%; the middle of this interval gives the computed average m. Moreover, under the name of each method in the graphics, we give the number of times that the best known solution value (obtained from these trials or from others with greater CPU times) has been found by the method during the N trials. Notice that the studied methods are ranked in the graphics according to their results: the method with the best average is on the left, while the one with the worst average (except the methods which do not appear in the graphics) is on the right.

All the tests have been done on a Sun station (SunOS Ultra 5, Unix Solaris 5.9, 500 MHz, memory of 128 Mo) with a program written in C language.

Before detailing the results, we recall the names of the 19 methods that we studied in Table 1. Remember that LB is in fact a simulated annealing and that the methods TB, TD and TA come from the threshold accepting method.

For the graphs that we describe now, we do not know the optimal value with an absolute certainty. But experiments done with much greater CPU times (up to several days) show that what we call "best known value" below seems to be in fact the minimum value.

4.1. *rand100-100*

The graph *rand100-100* has 100 vertices; the weights of its edges are randomly chosen between −100 and 100 with a uniform law. The CPU time given to each method is 7 s. The best known solution value is −24 296. For this graph (and only for this one), 1000 trials have been performed (Fig. 4, Table 2).

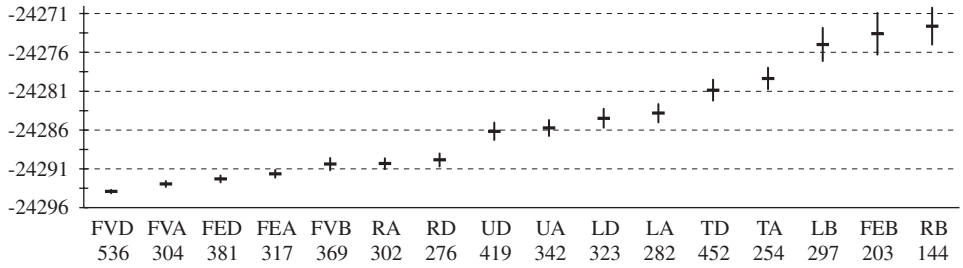

Fig. 4. Results for *rand100-100*.

Table 2
Results of TB, UB and Q for *rand100-100*

Other results	TB	UB	Q
Average	−24 248	−24 221	−23 906.5
Confidence interval at 95%	[−24 251, −24 245]	[−24 225, −24 217]	[−23 915, −23 898]
Best found value	−24 296 (98 times)	−24 296 (62 times)	−24 296 (1 time)

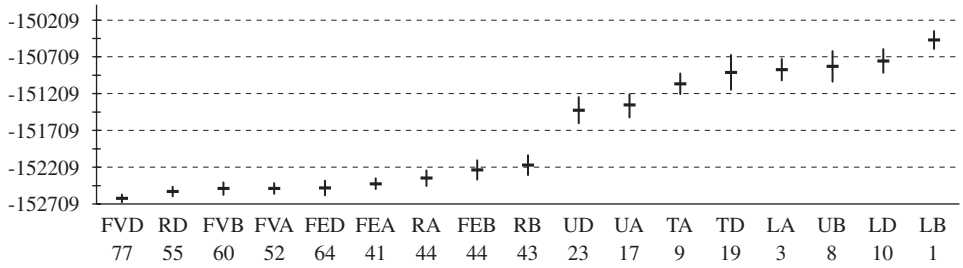

Fig. 5. Results for *rand300-100*.

Table 3
Results of TB and Q for *rand300-100*

Other results	TB	Q
Average	−149 216	−145 202
Confidence interval at 95%	[−149 435, −148 996]	[−145 421, −144 983]
Best found value	−152 159	−149 444

4.2. rand300-100

The graph *rand300-100* has 300 vertices; the weights of its edges are randomly chosen between −100 and 100 with a uniform law. The CPU time given to each method is 40 s. The best known solution value is −152 709 (Fig. 5, Table 3).

4.3. rand500-100

The graph *rand500-100* has 500 vertices; the weights of its edges are randomly chosen between −100 and 100 with a uniform law. The CPU time given to each method is 65 s. The best known solution value is −309 125 (Fig. 6, Table 4).

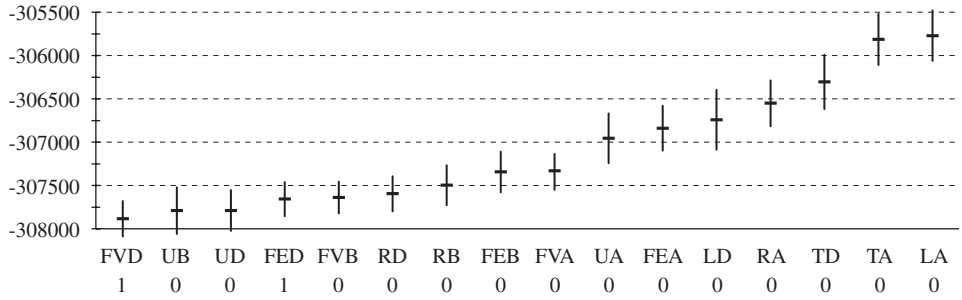

Fig. 6. Results for *rand500-100*.

Table 4
Results of LB, TB and Q for *rand500-100*

Other results	LB	TB	Q
Average	−304 220	−298 683	−288 401
Confidence interval at 95%	[−304 451, −303 989]	[−299 242, −298 124]	[−288 766, −288 036]
Best found value	−306 932	−304 450	−292 985

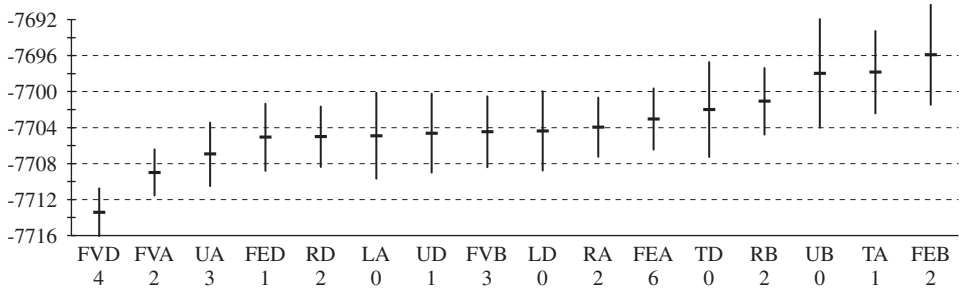

Fig. 7. Results for *rand300-5*.

Table 5
Results of LB, TB and Q for *rand300-5*

Other results	LB	TB	Q
Average	−7685.2	−7579.1	−7305.4
Confidence interval at 95%	[−7693, −7677.4]	[−7592.4, −7565.8]	[−7315.3, −7295.5]
Best found value	−7717	−7729	−7449

4.4. rand300-5

The graph *rand300-5* has 300 vertices; the weights of its edges are randomly chosen between −5 and 5 with a uniform law. The CPU time given to each method is 40 s. The best known solution value is −7732 (Fig. 7, Table 5)

4.5. zahn300

The graph *zahn300* has 300 vertices; the weights of its edges are randomly chosen in the set {−1, 1} with a uniform law; thus this graph is an instance of the problem of Zahn [35]. The CPU time given to each method is 50 s. The best known solution value is −2504 (Fig. 8, Table 6).

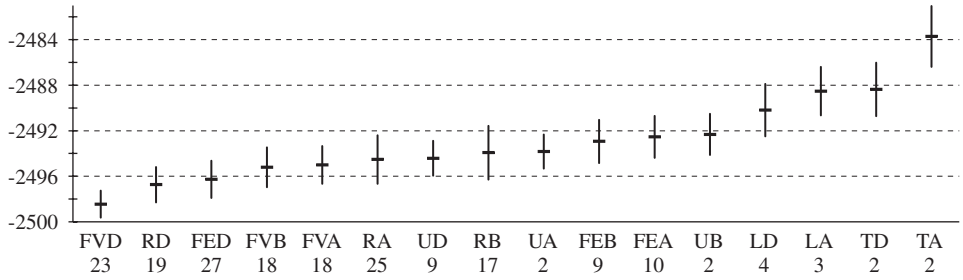

Fig. 8. Results for *zahn300*.

Table 6
Results of LB, TB and Q for *zahn300*

Other results	LB	TB	Q
Average	−2479.8	−2421.1	−2341.35
Confidence interval at 95%	[−2482.3, −2477.3]	[−2426.8, −2415.4]	[−2345, −2337.7]
Best found value	−2503	−2495	−2391

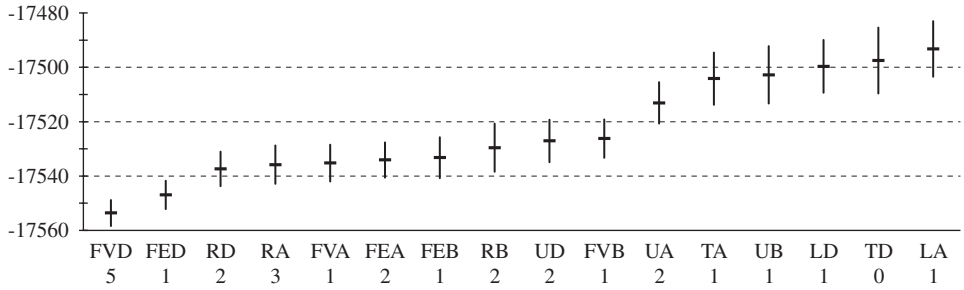

Fig. 9. Results for *sym300-50*.

Table 7
Results of LB, TB and Q for *sym300-50*

Other results	LB	TB	Q
Average	−17443.8	−17222.7	−16623
Confidence interval at 95%	[−17459.4, −17428.2]	[−17254.8, −17190.6]	[−16649, −16597]
Best found value	−17 586	−17 556	−16 942

4.6. sym300-50

The graph *sym300-50* has 300 vertices. To generate it, we first choose 50 symmetric relations defined on the vertices of *sym300-50*; for each relation, the probability for a given pair of vertices to be related is equal to 0.5; then the weight $w(x, y)$ of the edge $\{x, y\}$ is the difference between the number of relations for which x and y are not related and the number of relations for which they are related (thus the weights are between −50 and 50; see Section 2). The CPU time given to each method is 35 s. The best known solution value is −17 592 (Fig. 9, Table 7).

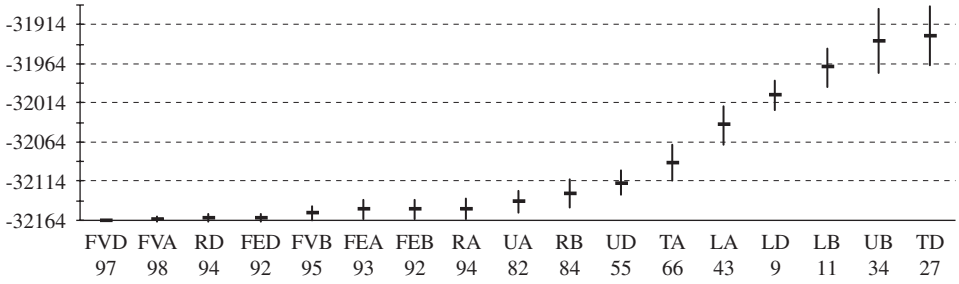

Fig. 10. Results for *régnier300-50*.

Table 8
Results of Q and TB for *régnier300-50*

Other results	Q	TB
Average	−31 796	−31 482
Confidence interval at 95%	[−31 833, −31 759]	[−31 549, −31 415]
Best found value	−32 164 (1 time)	−32 164 (3 times)

Table 9
Ranks of the 19 methods for the 7 graphs

	UB	UD	UA	LB	LD	LA	RB	RD	RA	TB	TD	TA	FVB	FVD	FVA	FEB	FED	FEA	Q
rand100-100	18	8	9	14	10	11	16	7	6	17	12	13	5	1	2	15	3	4	19
rand300-100	15	10	11	17	16	14	9	2	7	18	13	12	3	1	4	8	5	6	19
rand500-100	2	3	10	17	12	16	7	6	13	18	14	15	5	1	9	8	4	11	19
rand300-5	14	7	3	17	9	6	13	5	10	18	12	15	8	1	2	16	4	11	19
zahn300	12	7	9	17	13	14	8	2	6	18	15	16	4	1	5	10	3	11	19
sym300-50	13	9	11	17	14	16	8	3	4	18	15	12	10	1	5	7	2	6	19
régnier300-50	16	11	9	15	14	13	10	3	8	19	17	12	5	1	2	7	4	6	18

4.7. régnier300-50

The graph *régnier300-50* has 300 vertices. To generate it, we first choose 50 bipartitions defined on the vertices of *régnier300-50*; for each bipartition, the probability for a given vertex to be in the first cluster of the bipartition is equal to the probability to be in the other cluster of the bipartition (and thus is equal to 0.5); as for *sym300-50*, the weight $w(x, y)$ of the edge $\{x, y\}$ is the difference between the number of bipartitions for which x and y are not together and the number of bipartitions for which they are together (thus the weights are still between −50 and 50); this problem is an instance of the problem of Régnier. The CPU time given to each method is 20 s. The best known solution value is −32 164 (Fig. 10, Table 8).

5. Analysis of the results

Table 9 shows the rank of each method in the seven orders specified above.

From these results, it appears that the average value provided by FVD is always the best and that the method is very steady: its standard deviation is the smallest (in other words, the length of the 95% confidence interval is the smallest). Moreover, it always found what seems to be an optimal solution during the trials (even if the frequency of finding such a solution may be greater for other variants, depending on the considered graph), which is not always the case for LB (and for some other variants). For the other methods, the situation is not so simple: their ranks are not always the same.

Table 10
Borda scores of the 19 methods

	UB	UD	UA	LB	LD	LA	RB	RD	RA	TB	TD	TA	FVB	FVD	FVA	FEB	FED	FEA	Q
Borda scores	90	55	62	114	88	90	71	28	54	126	98	95	40	7	29	71	25	55	132

Thus, in order to be able to compare them, we apply some classic methods of ordinal data analysis. There are different ways to aggregate m linear orders O_k $(1 \leqslant k \leqslant m)$ into a linear order O so that O summarizes the orders O_k $(1 \leqslant k \leqslant m)$ as well as possible (see for instance [3]). We successively apply three procedures, well-established and well-known in the theory of ordinal aggregation for their axiomatic properties, respectively, due to Borda [5] and Kemeny [25] (in fact, it seems that it was already suggested by Condorcet [12]), or derived from the one proposed by Copeland [13].

5.1. Borda's procedure

In the procedure of Borda, a method ranked at the kth place in an order is given k points. Then, for each method, we compute the total number that it receives; this number is the *Borda score* of the method; with respect to the values reported in Table 9, the Borda score of a method is the sum of the values located in the column associated with this method. Finally, we rank the methods according to the increasing Borda scores; the linear orders that we obtain are the solutions O (with respect to this procedure) that we look for. Here, the Borda scores of the 19 methods are displayed in Table 10.

Thus, the orders provided by Borda's procedure are the linear extensions of the following total preorder:

FVD>FED>RD>FVA>FVB>RA>(FEA=UD)>UA>(FEB=RB)>LD>(LA=UB)>TD>TA>LB>TB>Q.

5.2. Copeland's procedure

Let μ and μ' be two methods, and let $m_{\mu\mu'}$ be the number of orders in which μ is placed before μ'. For each method μ, we compute the quantity $\sigma(\mu)$ defined by $\sigma(\mu) = \sum_{\mu'}(m_{\mu\mu'} - m_{\mu'\mu})$. Then, we rank the methods according to the decreasing values of σ. Here, we obtain the linear orders which are the linear extensions of the following total preorder:

FVD>FED>RD>FVA>FVB > RA>(FEA=UD)>UA>(FEB = RB)>LD>(LA=UB)>TA>TD>LB>TB>Q.

This preorder is the same as the one found by Borda's procedure, except that TA and TD do not appear in the same order.

5.3. Kemeny's procedure

Another way to aggregate linear orders consists in looking for a linear order which minimizes the total number of disagreements with respect to the given orders (as in Section 2, this number of disagreements is given by the symmetric difference distance). So, in a way, this provides the best compromise of the individual rankings in order to obtain a global ranking. This procedure leads here to three linear orders, which differ only by the places assigned to FVA, FED and RD:

FVD>FVA>FED>RD>FVB>FEA>RA>UD>FEB>RB>UA>LD>TA>LA>UB>TD>LB>TB>Q,
FVD>FED>RD>FVA>FVB>FEA>RA>UD>FEB>RB>UA>LD>TA>LA>UB>TD>LB>TB>Q,
FVD>RD>FVA>FED>FVB>FEA>RA>UD>FEB>RB>UA>LD>TA>LA>UB>TD>LB>TB>Q.

There is a great similarity between the ranks of the 19 methods over the previous three procedures: these ranks are not exactly the same, but almost. If we consider a less ordinal point of view and if we pay attention to the values provided by the 19 methods, we may notice that all the variants lead to solutions with very near values, which is not surprising since a method like LB (simulated annealing) already gives very good solutions; but on the other hand, it is well-known that it is quite difficult to improve already good solutions, as those found by simulated annealing. Thus, to measure such an improvement (if any), let $\overline{f_M}$ be the average value provided by method M (with $M \in$ {FVB, FVD, FVA, FEB, FED, FEA, RB, RD, RA, UB, UD, UA, LB, LD, LA, TB, TD, TA, Q}) for a given graph; Table 11 shows the values $(\overline{f_M} - \overline{f_{\mathrm{LB}}})/(\overline{f_{\mathrm{LB}}} - \overline{f_{\mathrm{Q}}})$ for $M \in$ {FVB, FVD, FVA, FEB, FED, FEA, RB, RD,

Table 11
Improvements brought to LB by each method with respect to the gap between Q and LB, in percentage

M	*rand100-100*	*rand300-100*	*rand500-100*	*rand300-5*	*zahn300*	*sym300-50*	*régnier300-50*
FVD	5.2	41	23	7.5	14	13	114
FED	4.7	38	22	5.3	12	13	112
RD	4.1	39	21	5.2	12	11	112
FVA	4.9	38	20	6.3	11	11	113
FVB	4.2	38	22	5.1	11	10	108
FEA	4.5	37	17	4.7	9	11	106
RA	4.2	36	15	4.9	11	11	105
UD	3.1	18	23	5.1	11	10	86
FEB	−0.3	34	20	2.9	9	11	105
RB	−0.6	34	21	4.2	10	10	94
UA	2.9	17	17	5.7	10	8	100
LD	2.6	5	16	5.1	8	7	21
TA	1.2	11	10	3.3	3	7	71
LA	2.4	8	10	5.2	6	6	43
UB	−14.6	7	23	3.4	9	7	−19
TD	1.6	8	13	4.4	6	6.5	−23
LB	0	0	0	0	0	0	0
TB	−7.3	−24	−35	−28	−43	−27	−283

RA, UB, UD, UA, LB, LD, LA, TB, TD, TA}, expressed in percentage. In other words, Table 11 measures the improvement brought by method *M* to simulated annealing (LB) when we use the improvement brought by LB to repeated descents (Q) as the unit.

From these experiments, we may draw the following global remarks (they hold a priori for the studied instances, but they are strengthened by other instances of the same problem not reported here; anyway, it would be unwise to conclude that they would stand for every combinatorial optimization problem; moreover, even for this problem, remember that the studied instances are randomly generated, and maybe the behavior of some methods would be slightly different if applied to another kind of instances; anyway, we think that the following conclusions remain valid in general):

- the "forgotten vertices" method FVD is always the best (in these experiments) and it always finds what seems to be an optimal solution at least once; one possible explanation of this phenomenon is that more iterations of this variant can be run (and thus more solutions can be evaluated) because the neighborhood is only partially explored and therefore more neighbors can be scanned, and because computations are faster since less data need to be considered;
- the variants FV, FE, R and, to a lower extent, U of the noising methods behave well with respect to simulated annealing (especially when there is no descent inserted, as in LB) or to the threshold accepting methods, which anyway provide very good results;
- variants D and A are rather better than the variant B: this shows that it is almost always useful to insert descents during the process;
- variant A provides very good solutions: it is a strong incentive for paying attention to such a way to tune the parameters of metaheuristics automatically;
- still globally, the ranking of the noise-adding-schemes is: FV > FE > R > U > L > T > Q. More precisely, FV is the best, FE and R are rather similar and much better than the others (except FV), U is rather Good, L and T appear as (relatively) rather bad (especially when descents are not inserted, i.e. TB), and Q is far away.

6. Conclusion

From this work, it appears that, for the studied clique partitioning of a weighted graph, it can be worth trying several noise-adding-schemes. We do not present here results computed when the noise rate does not decrease down to 0; nevertheless, the efficiency of some methods when applied to some instances would be improved if the noise rate is not decreased down to 0 (as it was the case for the experiments done on the Traveling Salesman Problem in [9]).

We may notice the good results provided here by the automatic variant of the studied noise-adding-schemes: they are not far from those obtained by the corresponding D-methods. The main advantage of this automatic version is that it is not necessary to spend time to find a good tuning of the usual parameters: here, the user has just to give the CPU time that he or she wishes to spend to solve his or her problem, and the algorithm computes proper parameters of the noising method in the same time as it computes the solution itself.

More generally, the variants developed in this paper may provide better results than some other approaches used to tackle hard problems, like simulated annealing (or the threshold accepting algorithm), though this method is already very good. Of course, because of the efficiency of simulated annealing, the improvements brought by FVD (the "forgotten vertices with descents" method) or by the other variants to LB (a classic simulated annealing but with a cyclic exploration of the neighborhood) are not very high. But, because it is essential in some fields like genomics to be able to get always closer to the optimal solution than what can be currently done, even if it is only by a little bit, we think that the search and the study of more efficient variants deserve interest. In future, such a search will remain one of our motivations for looking for other variants of the noising methods and, more generally, of metaheuristics.

References

[1] E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization. Wiley, New York, 1997.
[2] J.-P. Barthélemy, B. Leclerc, The median procedure for partitions, in: I. Cox, P. Hansen, B. Julesz (Eds.), Partitioning Data Sets, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 19, 1995, pp. 3–34.
[3] J.-P. Barthélemy, B. Monjardet, The median procedure in cluster analysis and social choice theory, Math. Soc. Sci. 1 (1981) 235–267.
[4] J.-P. Barthélemy, B. Monjardet, The median procedure in data analysis: new results and open problems, in: H.H. Bock (Ed.), Classification and Related Methods of Data Analysis, North-Holland, Amsterdam, 1988, pp. 309–316.
[5] J.-C. Borda, Mémoire sur les élections au scrutin, Histoire de l'Académie royale des Sciences pour 1781, Paris, 1784.
[6] S.A. Canuto, M.G.C. Resende, C.C. Ribeiro, Local search with perturbations for the prize-collecting Steiner tree problem in graphs, Networks 38 (2001) 50–58; a preliminary version appeared in the Proceedings of the Third Metaheuristics International Conference, Angra dos Reis, Brazil, 1999, pp. 115–119.
[7] I. Charon, O. Hudry, The noising method: a new combinatorial optimization method, Oper. Res. Lett. 14 (1993) 133–137.
[8] I. Charon, O. Hudry, Variations on the noising schemes for a clustering problem, Proceedings of the Third Metaheuristics International Conference, Angra dos Reis, Brazil, 1999, pp. 147–150.
[9] I. Charon, O. Hudry, Application of the noising method to the Travelling Salesman Problem, European J. Oper. Res. 125 (2000) 266–277.
[10] I. Charon, O. Hudry, The noising methods: a survey, in: P. Hansen, C.C. Ribeiro (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Dordrecht, 2001, pp. 245–262.
[11] I. Charon, O. Hudry, Self-tuning of the noising methods, submitted for publication.
[12] M.J.A.N. Condorcet, Caritat, marquis de Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix, Paris, 1785.
[13] A.H. Copeland, A 'reasonable' social welfare function, Seminar on Applications of Mathematics to Social Sciences, University of Michigan, USA, 1951.
[14] S.G. De Amorim, J.-P. Barthélemy, C.C. Ribeiro, Clustering and clique partitioning: simulated annealing and tabu search approaches, J. Classification 9 (1992) 17–42.
[15] K.A. Dowsland, Simulated annealing, in: C.R. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, London, 1995, pp. 20–69.
[16] G. Dueck, T. Scheurer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, J. Comput. Phys. 90 (1990) 161–175.
[17] G. Dueck, J. Wirsching, Threshold accepting algorithms for multi-constraint 0-1 knapsack problems, Technical paper TR 89 10 016, IBM Heidelberg Scientific Center, Heidelberg, 1989.
[18] M.B. Eisen, P.T. Spellman, P.O. Brown, D. Botstein, Cluster analysis and display of genome-wide expression patterns, Proc. Natl. Acad. Sci. USA 95 (25) (1998) 14863–14868.
[19] F.W. Glover, G.A. Kochenberger, Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, 2003.
[20] A. Guénoche, F. Denizot, Stratégie de séquençage par ordonnancement de contigs, submitted for publication.
[21] D. Guillaume, F. Murtagh, An application of XML and XLink using a graph-partitioning method and a density map for information retrieval and knowledge discovery, in: D.M. Mehringer, R.L. Plante, D.A. Roberts (Eds.), Astronomical Data Analysis Software and Systems VIII, ASP Conference Series, vol. 172, San Francisco, 1999, p. 278.
[22] D. Guillaume, F. Murtagh, Clustering of XML documents, Comput. Phys. Commun. 127 (2000) 215–227.
[23] P. Hansen, B. Jaumard, Cluster analysis and mathematical programming, Math. Programming 79 (1997) 191–215.
[24] P. Hansen, C.C. Ribeiro (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Boston, 2001.
[25] J.G. Kemeny, Mathematics without numbers, Daedalus 88 (1959) 577–591.
[26] M. Krivanek, J. Moravek, NP-hard problems in hierarchical-tree clustering, Acta Informatica 23 (1986) 311–323.
[27] G. Laporte, I.H. Osman Metaheuristics: a bibliography, Ann. Oper. Res. 63 (1996) 513–623.
[28] I.H. Osman, J.P. Kelly (Eds.), Meta-heuristics: Theory and Applications, Kluwer Academic Publishers, Boston, 1996.

[29] C.R. Reeves (Ed.) Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, London, 1995.
[30] S. Régnier, Sur quelques aspects mathématiques des problèmes de classification automatique, I.C.C. Bull. 4 (1965) 85–111.
[31] V. Sudhakar, C. Siva Ram Murthy, A modified noising algorithm for the graph partitioning problem, Integration VLSI J. 22 (1997) 101–113.
[32] S. Voss, S. Martello, I.H. Osman, C. Roucairol, (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1998.
[33] Y. Wakabayashi, Aggregation of binary relations: algorithmic and polyhedral investigations, Ph.D. Thesis, Augsbourg, 1986.
[34] Y. Wakabayashi, The Complexity of Computing Medians of Relations, Resenhas 3 (3) (1998) 323–349.
[35] C.T. Zahn, Approximating symmetric relations by equivalence relations, SIAM J. Appl. Math. 12 (1964) 840–847.